“PUBLISH-SUBSCRIBE BASED COMMUNICATION MODEL”

Submitted by:
Nitesh Agarwal
(110ei0484)

Under the guidance of
Prof. Upendra Kumar Sahoo

National Institute of Technology, Rourkela
Orissa-769008

May-2014

ACKNOWLEDGEMENTS

On the submission of my project report of ***"Publish Subscribe based communication model"***, I would like to extend my gratitude & sincere thanks to my supervisor Prof. U. K. Sahoo, Professor, Department of Electronics and Communication Engineering for his support during the course of my work.

Nitesh Agarwal

CERTIFICATE

This is to certify that the work in this project entitled ***"Publish Subscribe based communication model"*** by ***Nitesh Agarwal***, has been carried out under my supervision in partial fulfillment of the requirements for the degree of Bachelor of Technology in **'Electronics and Instrumentation Engineering'** during the session 2010 – 2014 in the Department of Electronics and Communication Engineering, National Institute of Technology, Rourkela and this work has not been submitted elsewhere for a degree.

Place :
Date :

Prof. U. K. Sahoo
Professor, Dept. of ECE
National Institute of Technology, Rourkela

ABSTARCT

Publish-Subscribe is an expression used to describe an application model in which supplier of some data (Publisher) is decoupled from the consumers of that data (Subscriber). Publish-Subscribe paradigm is an asynchronous messaging paradigm. Publishers are unaware of receivers i.e. they do not send messages to specific receivers. Instead, the messages published are divided into classes, oblivious of the subscribers. Subscribers express interest in one or more classes, and only receive messages that are of interest, without any awareness of the publishers. The thesis studies publish-subscribe model and the paradigm's advantages in **distributed computing.** The thesis also proposes a scheme to respond to user's interests derived from both published content and subscribed content, if any. This model has applications in Blogging Systems, Advertisements and Recommendation Systems.

CONTENTS

Chapter-1
INTRODUCTION

1.1 INTRODUCTION:

According to Google the web is made up of 60 trillion individual pages and it's constantly growing. Any user is interested in a minute fraction of this ever growing information. Conventionally users poll sources for information. However polling too frequently is inefficacious and polling less often may escape important updates. The data dissemination problem is made worse by the scale of internet. It becomes very important to have scalable model for applications related to the data dissemination problem with millions of interested users.

Publish-Subscribe uses push technology for data dissemination. It proactively pushes updates to users with matching interest. This makes for scalable, distributed applications.

Applications:

Personal -RSS feeds, online auctions,

Financial -Portfolio monitoring and management,

Security -Network anomalies, distributing software patches.

In general a Publish-Subscribe system consists of several nodes connected in a network called brokers. Subscribers associates themselves to brokers based on different criterion. Sources or Publishers publish information or generate data updates to a database, which may be located at a server, or distributed servers, or distributed nodes in the network. The notification could affect a

number of subscriptions. The Publish-Subscribe middleware ensures that the notification is distributed to all brokers hosting affected subscription.

1.2 PUBLISH-SUBSCRIBE

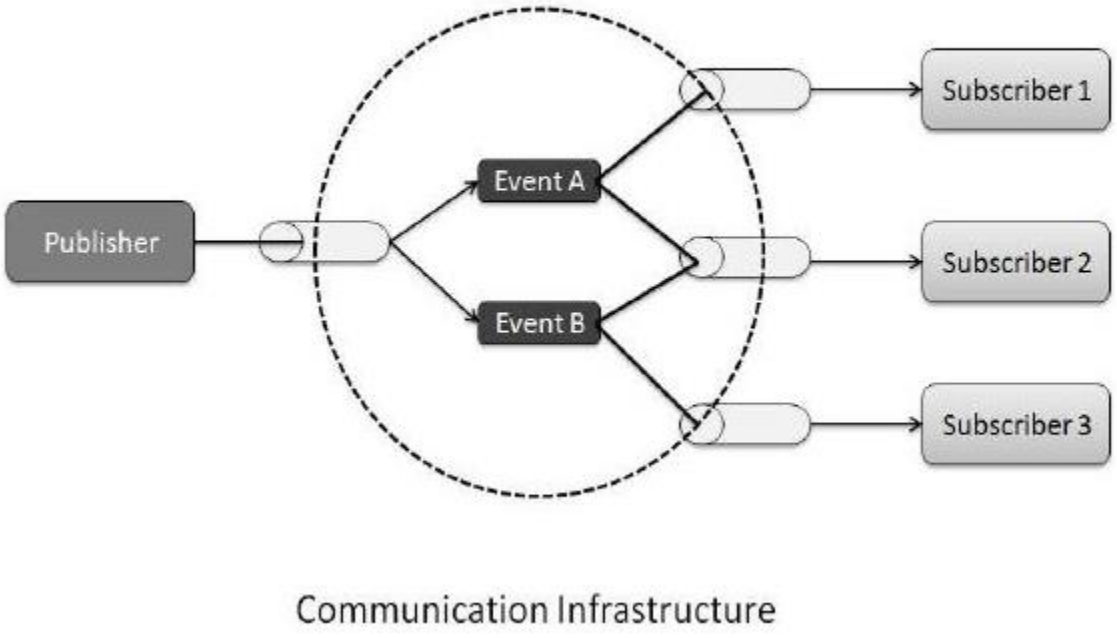

Fig. 1

Publish Subscribe is a term used to define an application model in which provider of some information (Publisher) is decoupled from the consumers of that information (Subscriber). Publish-Subscribe paradigm is an asynchronous messaging paradigm.

Publishers of messages are not programmed to send their messages to specific receivers. Rather, published messages are characterized into classes, without knowledge of what subscribers there may be.

Subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of what publishers there are.

With ever increasing size of the internet (data sources) e.g. RSS feeds, network monitoring log, etc. as well as refined data demand, data management has become critical in distributed systems. Users may want updates to be transformed, correlated, and aggregated.

Chapter-2
Berkeley Sockets

2.1 Introduction

Berkeley sockets (or BSD sockets) is a computing library with an application programming interface (API) for internet sockets and Unix domain sockets, used for inter-process communication (IPC).

2.2 Socket API

Socket API specifies how software components may interact with sockets.

2.2.1 Socket API functions:

- **socket()** : creates a new socket of a certain socket type.
- **bind()**: used on server side, associates a socket with socket address which comprises of socket address and local port number.
- **listen()**: used on server side, causes a bound TCP socket to enter listening state.
- **connect()**: used on client side, assigns a free local port number to a socket.
- **accept()**: used on server side, accepts received incoming attempt to create a new TCP connection from the remote client.

- **send()**, **recv()**, **write()**, **read()** are used for sending and receiving data to/from a remote socket.

2.2.2An example using socket api:

Following is a synchronous chat application using sockets in python3. Following is also an example of client-server communication model.

```
# server.py
# server binds a socket to
# a particular address and
# listen for client comm
from socket import *
HOST='127.0.0.1'
PORT=8000
s=socket(AF_INET,SOCK_STREAM)
s.bind((HOST,PORT))
s.listen(1) #how many connections it can recieve at one time
conn, addr=s.accept()
print("Connected by",addr)#print the address of the person connected
while True:
  data=conn.recv(1024).decode('UTF-8')
  if not data:
    break
  print("Recieved",repr(data))
  re=input("Reply:  ")
  if re=='':
    re=' '
  if re=='exit':
    break
  reply=bytes(re,'UTF-8')
  conn.sendall(reply)#sendall() sends to all nodes connected and send() sends
to one specific node
conn.close()
```

```
#client.py
# The client establishes communication
# with the server, sends a message and
```

```
# await a reply
from socket import *
s=socket(AF_INET,SOCK_STREAM)
s.connect(('127.0.0.1',8000))
while True:
  uinp=input("Your Message: ")
  if uinp=='':
    uinp=' '
  if uinp=='exit':
    break
  message=bytes(uinp,'UTF-8')
  s.send(message)
  print("Awaiting reply...")
  reply=s.recv(1024).decode('UTF-8')#1024 bytes is the max data that can be
recieved
  if not reply:
    break
  print("Received",repr(reply))
s.close()
```

In the above example PF_INET, SOCK_STREAM and IPPROTO_TCP are constants. Use of these constants creates a TCP socket.
Many other possibilities exist:

Domain	**Type**	**Protocol**
PF_UNIX, PF_INET, PF_INET6	SOCK_STREAM, SOCK_DGRAM	IPPROTO_TCP, IPPROTO_UDP

In the above example, server creates a socket, binds it to a particular address and listens for client communication. The client on the other hand contacts the server by connecting to the address that server socket is bound with and establishes a connection. A client may then send a message to the server and await reply.

In this type of communication the networking is coupled in space, time and synchronization. A server must exist for a client to communicate. Communication can only occur if both client and server are online.

TCP using sockets:

One socket on the server side is dedicated to waiting for a connection for each client that takes the initiative, a separate socket on the server side is created this is useful for all servers that must be able to serve several clients concurrently (web servers, mail servers, etc.).

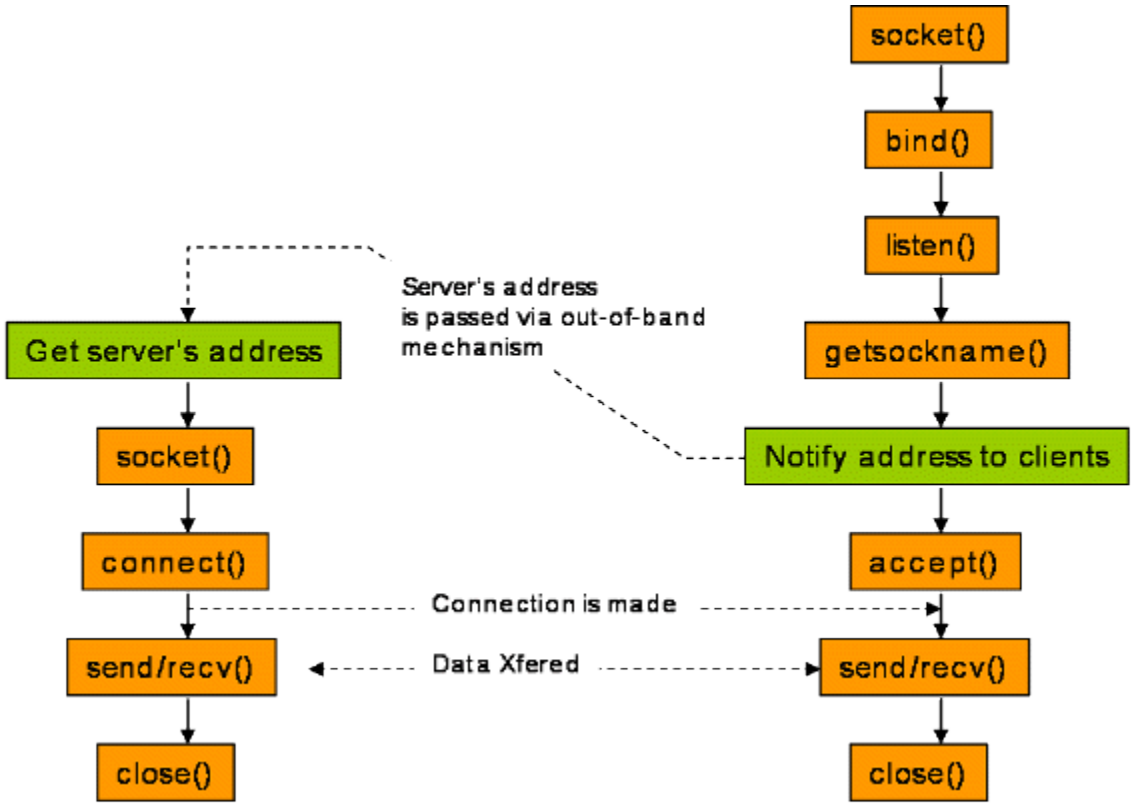

Fig. 2 Socket Communication

Chapter-3

Distributed Systems and Characteristics

3.1 Introduction

A Distributed System is a collection of independent computers that appears to its user as a single coherent system.

Distributed computing *is decentralized and parallel computing, using two or more computers communicating over a network to accomplish a common objective or task. The types of hardware, programming languages, operating systems and other resources may vary drastically.*

With respect to hardware nodes (machines) are autonomous and with respect to software the clients or users think they are communicating with a single system or entity.

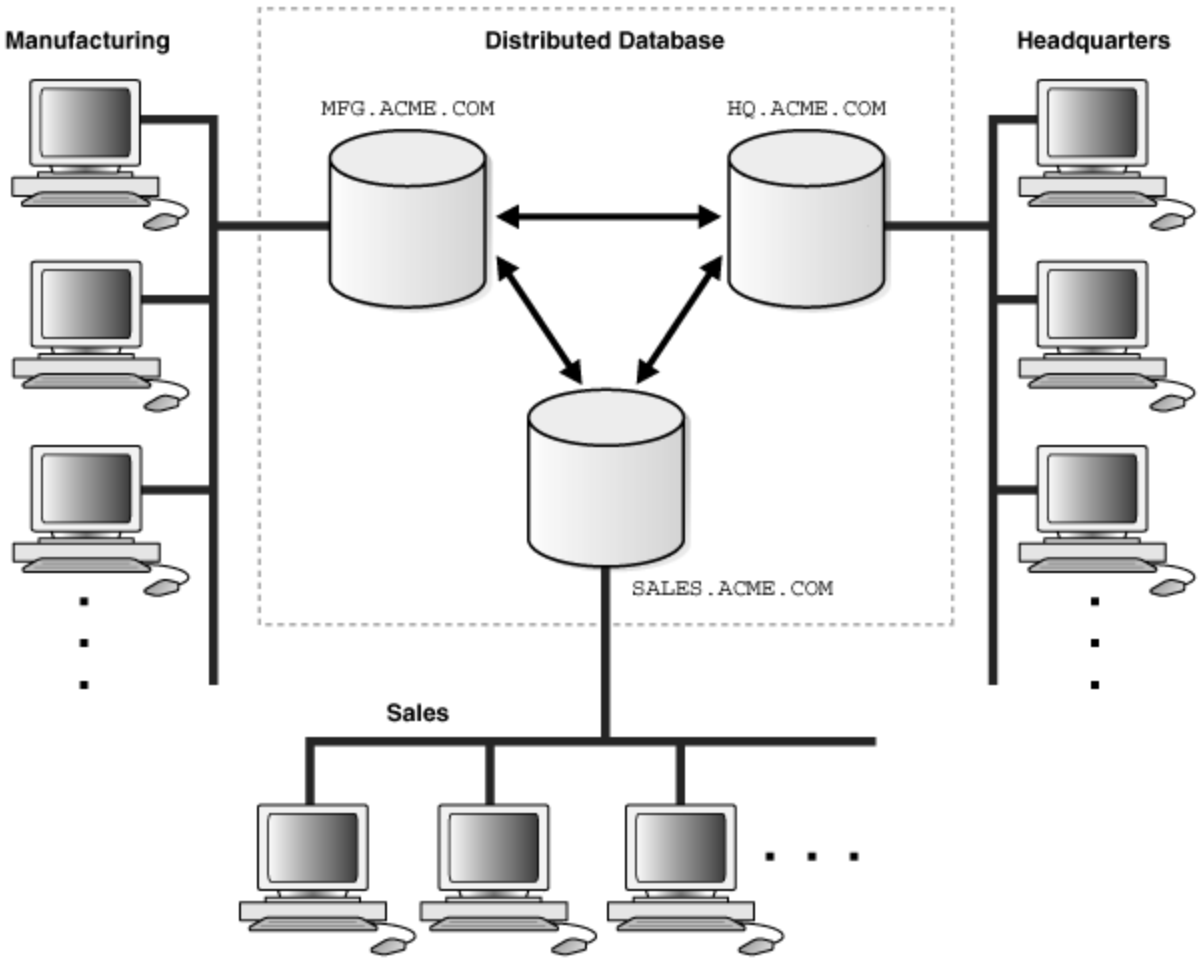

Fig. 3 Example of a Distributed System

3.2 Characteristics of Distributed Systems

Distributed System advocates addition of components or nodes without affecting existing nodes or the system as a whole. This improves performance, reliability, availability, fault-tolerance and scalability in sharing resources.

3.2.1 Key Characteristics of Distributed Systems:

1. Resource Sharing

2. Openness

3. Concurrency

4. Scalability

5. Fault Tolerance

6. Transparency

3.2.2 Resource Sharing:

Various resources may be shared usefully in a distributed system e.g. printers, disks, cd-roms, etc. or data. A resource manager is a software module that manages a set of resources of a particular type.

A resource sharing model describes how:

- Information and Services are made available,
- Resources or Services can be used,
- Resource provider and user communicate with each other.

3.2.3 Openness:

Openness relates to extensibility of the distributed system. Distributed systems can be extended in various ways- Hardware extensions, Software extensions. An open Distributed System can be extended and improved incrementally. A Distributed System is required to follow a uniform IPC (inter process communication) mechanism and publication of component interfaces (e.g. subject to standardization).

3.2.4 Concurrency

Components in Distributed Systems execute in concurrent process. Components access and update shared resources (e.g., variables, databases, device-drivers). Users and applications should be able to access shared data or objects without interference between them. Example: Transaction management, NFS, ATM (Automated Teller Machine). Integrity of the system maybe violated if concurrent updates are not coordinated. Retaining of integrity requires concurrency control where concurrent access to the same resource is synchronized.

3.2.5 Scalability

A system should be able to grow without affecting application algorithms. A system is scalable if it remains efficacious when there is significant increase in the amount of resources and number of users e.g. Internet.

Scalability denotes the ability of the system to handle an increasing future load. Requirements of scalability often lead to Distributed System architecture.

Challenges:

To control the cost of physical resources:

A system with k users is resource-scalable if the number of resources required to support them is of complexity O(k).

To control the performance loss (when the size of data increases):

A system is performance-scalable if the time it takes to access hierarchically ordered data is of complexity O(log k) where k is the size of data.

To prevent software resources from running out:

Dimension data structures such that the system can meet future requirements.

To avoid performance bottlenecks:

Requires decentralized algorithms (partitioning, caching and replication).

3.2.6 Fault Tolerance

Fault tolerance is the ability of a system to recover when failure occurs. It enables suppression of faults allowing user and application programs to complete their tasks despite the failure of hardware or software component. Fault tolerant design is based on two approaches:

- Hardware Redundancy
- Software Recovery

While Fault tolerance is required in some systems, it may as well be nuisance in some other because of the overhead it may cause.

Failures in distributed systems are partial. This makes error handling particularly difficult.

Techniques for handling failures:

- Detection of failure (e.g. checksum)
- Masking failures (retransmission in protocols, replication)
- Tolerating failures (as in web browsers)
- Recovery from failures
- Redundancy (replicate servers in failure-independent ways)

3.2.7 Transparency

The aspects of distributed systems/ distribution are made invisible to the client or the application user to provide a single centralized view of the system without worrying about the underlying design and implementation details of the system.

The distributed systems should be perceived as a single entity by the users or the application programmers rather than as a collection of autonomous systems, which are cooperating. The users should be unaware of where the services are located and also the transferring from a local machine to a remote one should also be transparent.

Types of Transparencies:

1. Access Transparency – Clients should be unaware of the distribution of the files. The files could be present on totally different set of servers which are physically distant and single set of operations should be provided to access these remote as well as the local files. E.g. NFS (network file systems), SQL queries and Navigation of Web.
2. Location Transparency-Clients should see a uniform file name space. Files or group of files may be relocated without changing their pathnames. A location transparent name contains no information about the named object's physical location.
3. Replication Transparency-For reliability a distributed file system may maintain replicated data at two or more sites. The clients should be

unaware of this. Example- Distributed Database Management System and mirroring of web pages.

4. Migration Transparency- This transparency allows the user to be unaware of the movement of information or processes within a system without affecting the operation of users and applications that are running. For e.g., Hot Swapping in Software.

   Hot Swapping refer to the ability to alter the running code of a program without needing to interrupt its execution.

   Interactive programming is a programming paradigm that uses extensive use of hot swapping, so the programming activity becomes part of the program flow itself.

5. Performance Transparency- Allows the system to be reconfigured to improve the performance as load varies. E.g. Hot Swapping.

Other Transparency includes Concurrency Transparency, Failure Transparency, and Scalability Transparency.

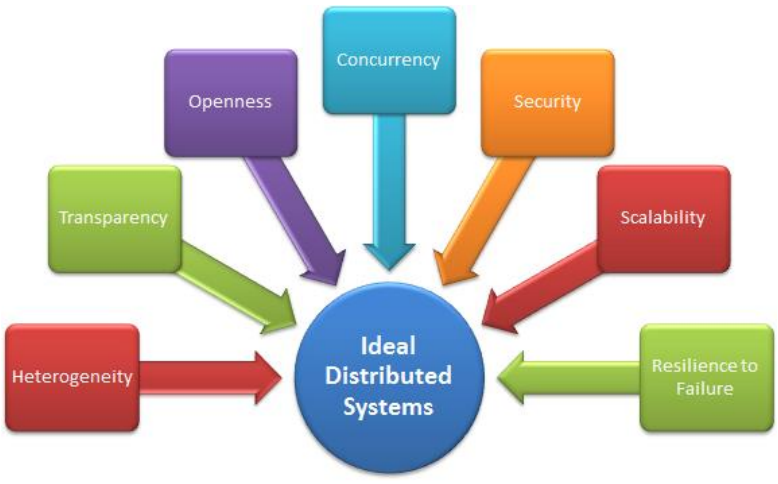

Fig. 4 Challenges of distributed systems

3.3 Distributed Systems Model

There are two interesting models for Distributed Systems:

- The Client-Server Model
- The Object Model

3.3.1 The client-server model:

The executing software is split into separate processes: *clients and servers.* Servers provide services and client uses these services. Clients usually initiate the communication with a server, whereas the server passively waits. Servers typically may serve many clients. The client must not be aware of the location of the servers. E.g. database servers (mysql-server), web servers (apache).

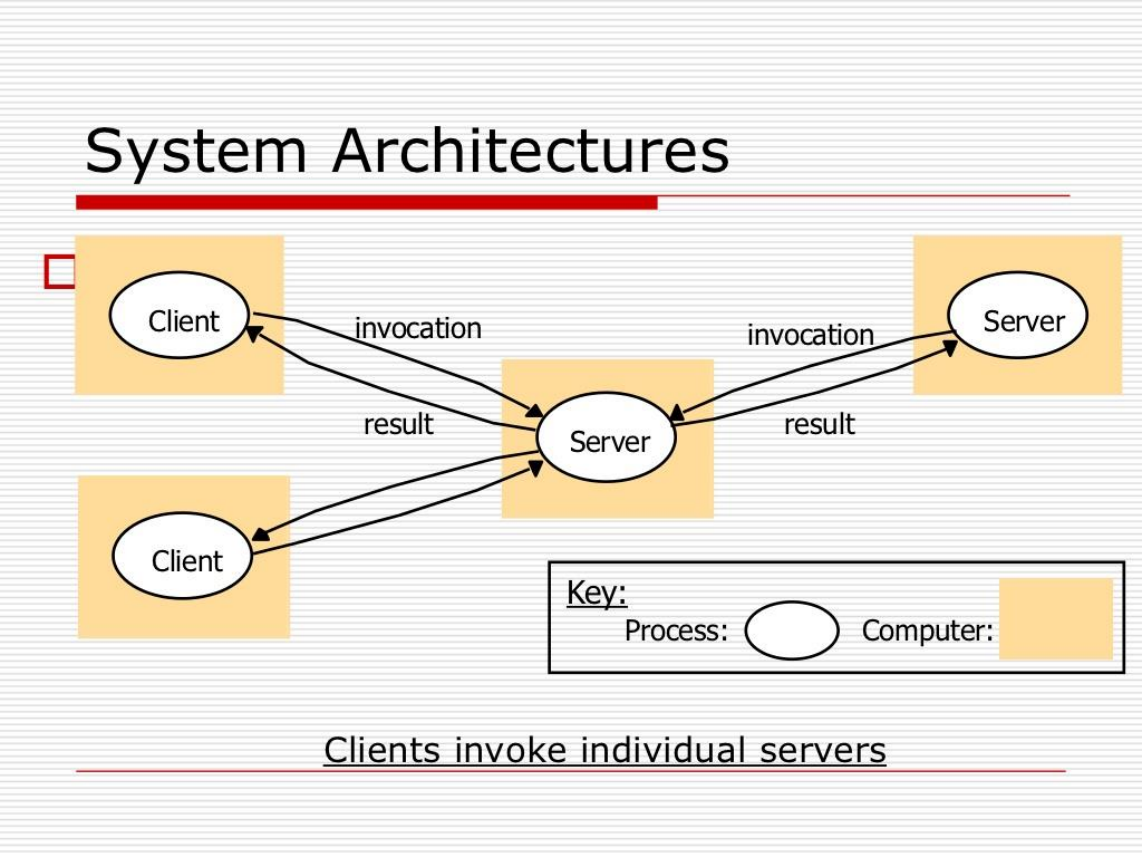

Fig. 5 Client-Server Architecture

3.3.2 The object model:

In the object model, every entity in a running program is viewed as an object with a message-handling interface providing access to its operations. In the object based model for distributed systems, each shared resource is an object. Objects are uniquely identified and may be moved anywhere in the network without changing their identities.

Distributed objects: The term distributed objects usually refer to software modules that are designed to work together, but reside either in multiple computers connected via a network or in different processes inside the same computer. One object sends a message to another object in a remote machine or process to perform some task. The results are sent back to the calling object.

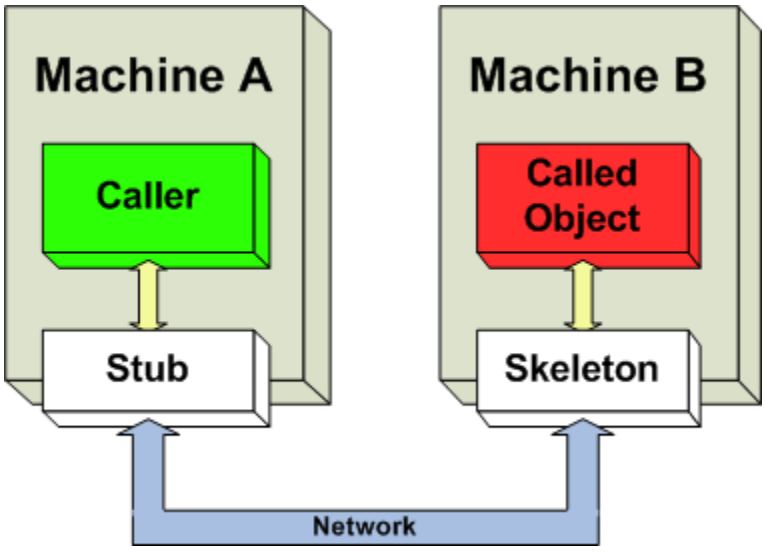

Fig. 6 A distributed object

3.3.3 Peer-to-Peer network architecture:

Peer-to-peer is a communications model in which each party has the same capabilities and either party can initiate a communication session. In some cases, peer-to-peer communications is implemented by giving each communication node both server and client capabilities. In recent usage, peer-to-peer has come to describe applications in which users can use the Internet to exchange files with each other directly or through a mediating server. In peer-to-peer (P2P) network, tasks (such as searching for files, streaming audio or video) are shared among multiple peers who each make a portion of their resources (processing power, disk storage and bandwidth) directly available to other network participants, without the need for centralized coordination by servers.

Peer-to-peer networks are used in many applications. The most commonly used application is file sharing which popularized the technology.

Applications : Instant messaging systems and online chat networks, Bitcoin (peer-to-peer based digital currency), etc.

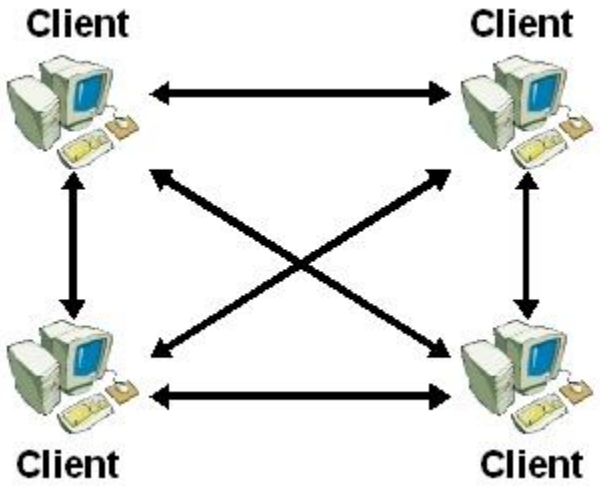

Fig. 7 peer-to-peer network

Chapter-4

PUBLISH-SUBSCRIBE AS DISTRIBUTED

4.1 Introduction

The intricacy of distributed systems has increased to a large extent over time. It is partly owed by the scale of Internet and its growth. Distributed Systems now involve thousands of entities, distributed all over the web whose location and behavior may vary throughout the lifetime of the system. These restrictions call for more flexible communication model and systems, reflecting the dynamic and decoupled nature of the applications. Peer to Peer synchronous communication leads to rigid and static applications.

Publish-Subscribe communication model advocates loosely coupled form of interaction required in scalable systems. It is event based interaction style that provides strong decoupling in time, space and synchronization.

4.2 How is Publish-Subscribe model well adapted to Distributed Environment?

Characteristics of distributed systems also manifested by Publish-Subscribe model.

- Multiple autonomous components- Autonomous components of Publish-Subscribe model are distributed notification system or distributed event broker, distributed database, publishers and subscribers.
- Components are not shared by all existing components in the system.
- Aspects of distribution are made invisible to the client. Event service provides this application in Publish-Subscribe model.
- Asynchronous nature of processes. Interaction between Publishers and Subscribers are asynchronous in nature. Both are completely oblivious of the existence of each other.
- Multiple points of control and failure.
- Space, time and synchronization decoupling makes the communication infrastructure well adapted to Distributed Systems.

4.3 Publish-Subscribe Interaction Scheme

The Publish-Subscribe interaction paradigm enables subscribers the facility to manifest their interest in an event or a pattern of events, in order to receive notification of an event produced by a Publisher that meet their registered interest. Producers publish information on a software bus (the event broker) and consumers subscribe to the information they want to receive from that bus.

This information is typically denoted by the term event and the term notification denotes the act of delivering it. The basic system model for Publish-Subscribe interaction relies on an event notification service providing storage and management for subscriptions and efficient delivery of events. Such an event service represents a neutral mediator between publishers, acting as producers of events, and subscribers, acting as consumer of events.

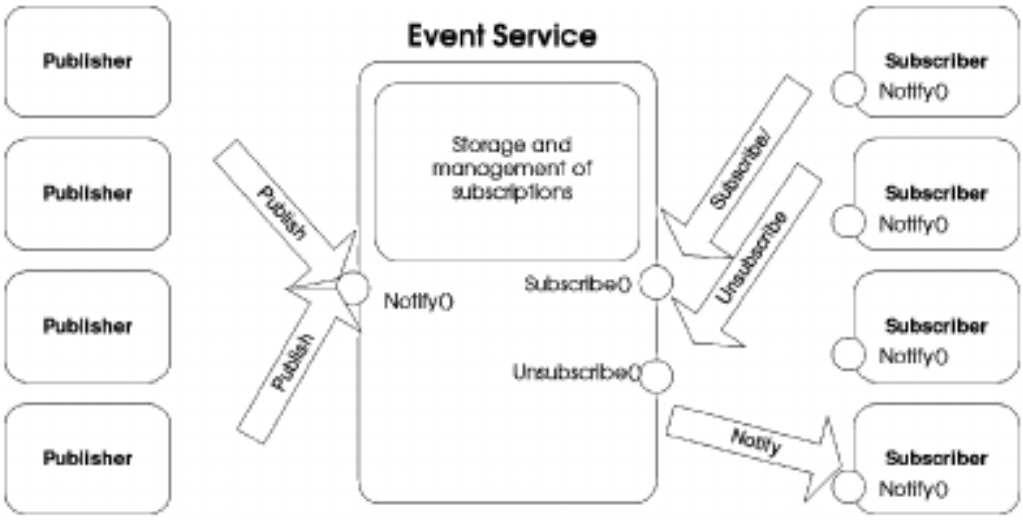

Fig. 8 A simple object based publish-subscribe system

4.4 Publish-Subscribe Event Broker

Subscribers log their interest in events usually by calling a subscribe() function on the event broker, without knowing the effective sources of these events. This subscription information remains stored in the event service and is not forwarded to publishers. The symmetric function unsubscribe() terminates a subscription.

To initiate an event, a publisher usually calls a publish() operation. The event broker propagates this event to all relevant subscribers. Every subscriber is notified of every event complying with its interest or rules. (though failures might prevent subscribers from receiving some events).

Publishers also often have the ability to advertise the nature of their future events through an advertise() operation. The provided information can be useful for (1) the event service to adjust itself to the expected flows of events, and (2) the subscribers to learn when a new type of information becomes available.

4.5 Space, Time and Synchronization Decoupling with Publish-Subscribe Paradigm

- **Space decoupling:** The communicating parties need not possess knowledge there existence. The publishers publish events through an event broker and the subscribers get these events indirectly through the event broker. The publishers do not usually hold references to the subscribers. They are also unaware of the number of subscribers participating in this communication. Similarly, Subscribers do not

usually hold references to the Publishers: neither do they know how many of these publishers are participating in the communication.

- **Time decoupling:** The communicating parties need not partake in the communication simultaneously. In particular, the publisher might publish some events while the subscriber is disconnected, and conversely, the subscriber might get notified about the occurrence of some event while the original publisher of the event is disconnected. The Publisher and Subscriber need not be online at the same time to enable communication. This is the most important advantage of using Publish-Subscribe model.

- **Synchronization decoupling:** Publishers are not blocked while producing events, and the occurrence of an event can get asynchronously notified (through a callback) to Subscribers, while performing some concurrent activity. The production and consumption of events do not happen in the main flow of control of the publishers and subscribers, and do not therefore happen in a synchronous manner.

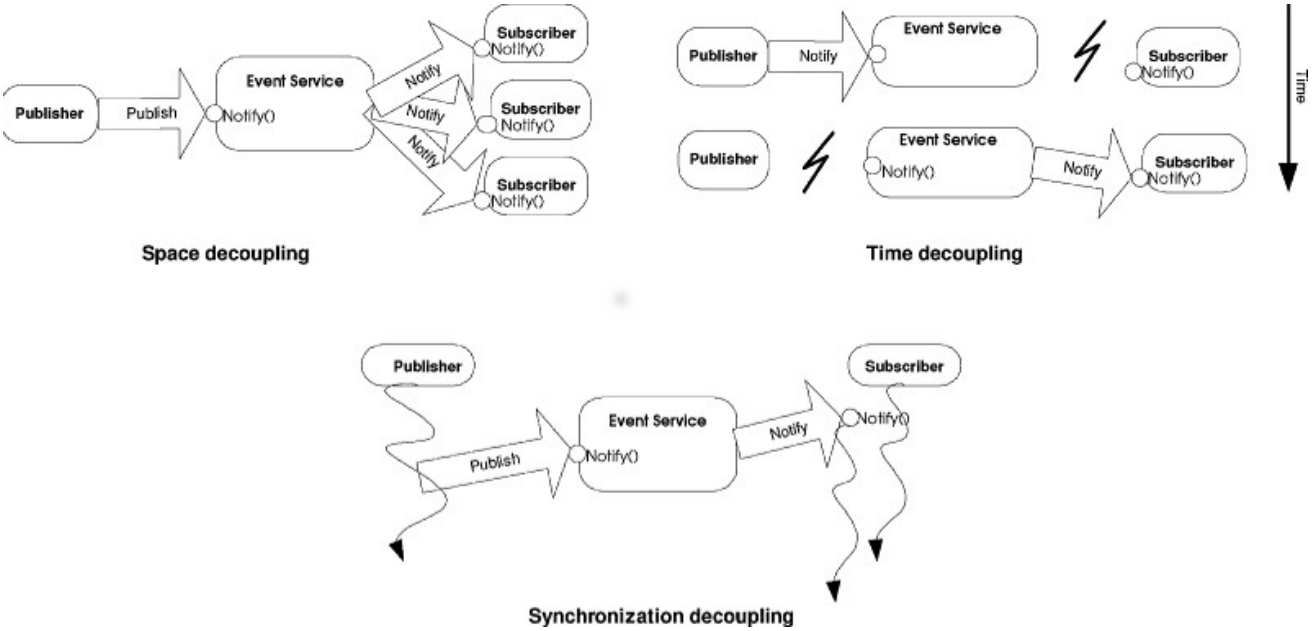

Fig. 9 Space, time and synchronization decoupling with publish-subscribe paradigm.

Chapter-5
TYPES OF PUBLISH-SUBSCRIBE

5.1 INTRODUCTION

Publish-Subscribe based Interaction paradigm consists of:

Publishers: generate information as events.

Subscribers: assert using subscriptions.

The set of published events is filtered by each subscription.

An *Event Broker* reports all published events which meets at least one of its subscriptions.

Subscriptions are issued by subscribers, by manifesting interests in specific events. A subscription is a constraint manifested on event schema. The event broker will notify an event e to a subscriber a if the values that define the event satisfy at least one of the subscription s issued by a. In this case we say e matches s.

From an abstract point of view the event schema defines an n-dimensional event space (where n is the number of attributes). In this space each event e represents a point. Each subscription s identifies a subspace. An event e matches the subscription s if and only if e is in the space defined by s.

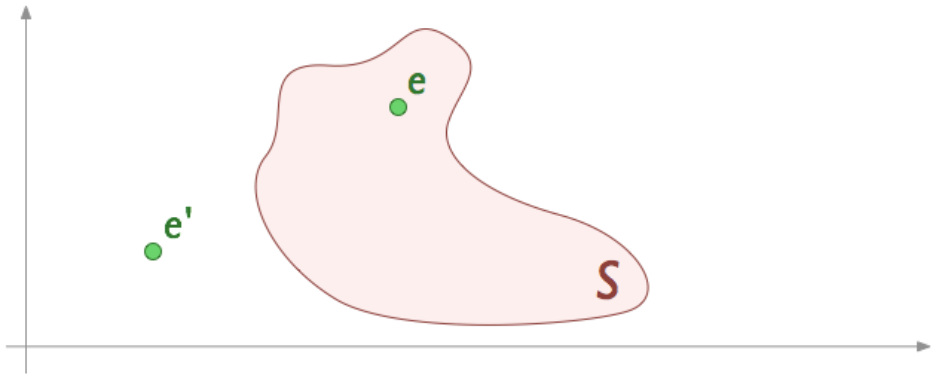

Fig. 10 Abstract Event Space

5.2 DIFFERENT TYPES OF PUBLISH-SUBSCRIBE

- Topic-based
- Hierarchy-based
- Content-based
- Type-based
- Concept-based
- XML-based

1. **Topic-based Publish-Subscribe**: Data published in the system is mostly unstructured, but each event is labeled with the identifier of a *topic* it is published in. Subscribers issue subscriptions containing the topics they are interested in.

   A topic can be thus represented as a *channel* connecting publishers to subscribers. For this reason the problem of data distribution in topic-based publish-subscribe systems is considered quite close to group communications.

Fig. 11 Topic Based Publish-Subscribe

2. **Hierarchy Based Publish-Subscribe:** Each event is labeled with the *topic* it is published in. The Subscribers emit subscriptions containing the topics they are interested in. Contrary to the previous model, here topics are organized in hierarchical structures which express a notion of confinement between topics. When a subscriber subscribes a topic, it will receive all the events published in that topic as well as in all the topics present in the corresponding sub-tree.

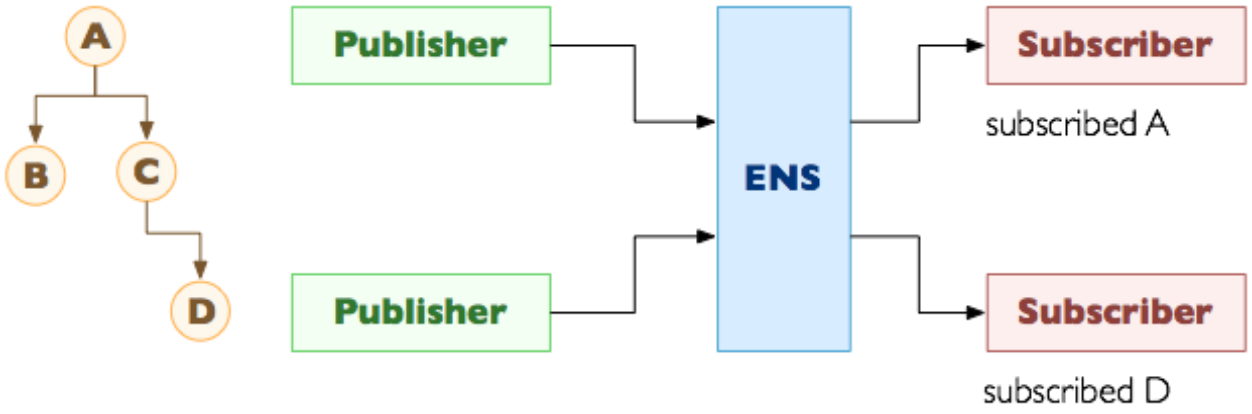

Fig. 12 Hierarchy Based Publish-Subscribe

3. **Content Based Publish-Subscribe:** All the data published in the system is mostly structured. Each subscription can be expressed as a conjunction of constrains expressed on attributes. The Event Notification Service filters out useless events before notifying a subscriber.

   Event1: {stock:"ipubmysub", value="10"}

   Event2: {stock:"ipubmysub", value="15"}

5.3 EVENT NOTIFICATION SERVICE (ENS) OR EVENT BROKER

The Event Broker is usually implemented as:

Centralized service: Event Broker is implemented on a single server.

Distributed service: Event Broker is constitutes a set of nodes, event brokers, which cooperate to implement the service.

To enable Scalability and Distribution Distributed Service is implemented.

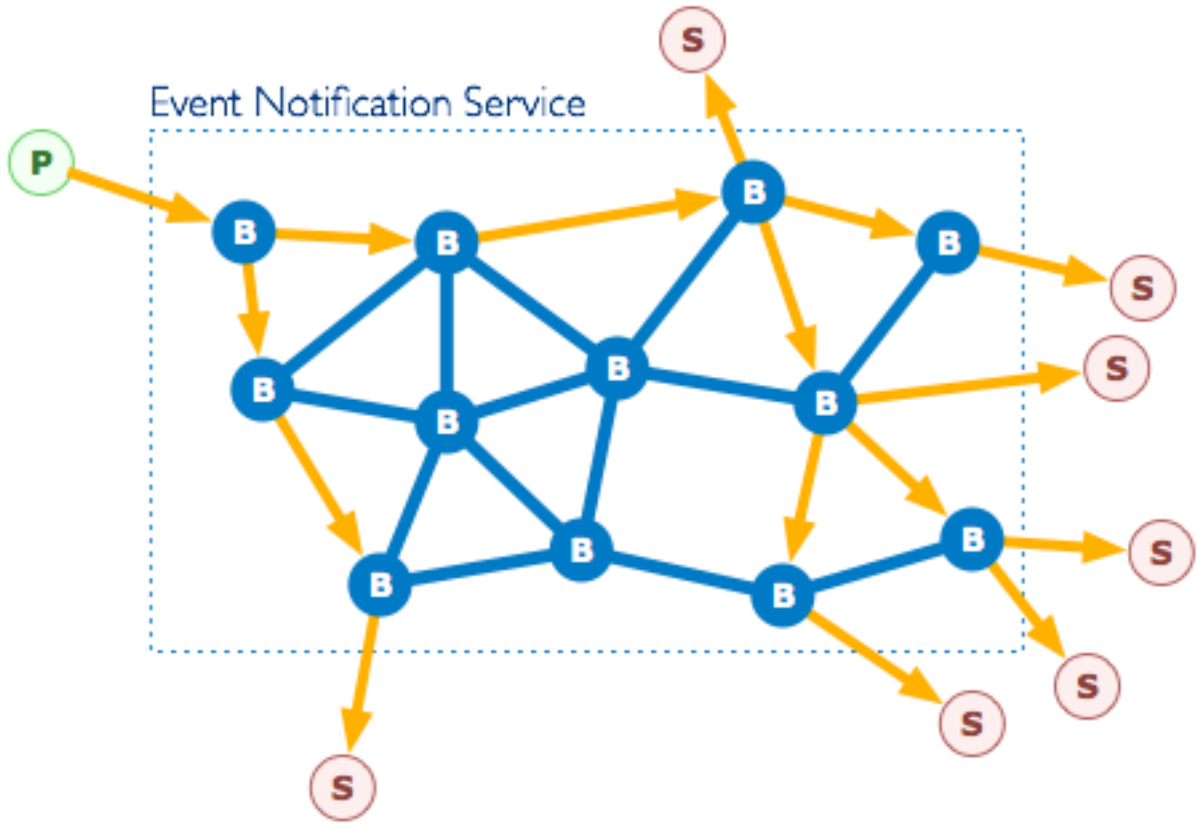

Fig. 13 Event Broker

5.3.1 Event Routing: An event routing mechanism routes each event inside the ENS from the broker where it is published to the broker(s) where it must be notified. This includes

- Filter-Based Routing
- Rendezvous Routing

5.3.1.1 **Filter-Based Routing:** Subscriptions are partially diffused in the system and used to build *routing tables. These tables, are then exploited during event diffusion to dynamically build a multicast tree that (hopefully) connects the publisher to all, and only, the interested subscribers.*

5.3.1.2 Rendezvous Routing: It is based on two functions, namely *Sub* and *Eve*, used to associate respectively subscriptions and events to brokers in the system.

- Given a subscription *s*, *Sub(s)* returns a set of brokers that store *s* and forward received events, matching *s* to all subscribers that subscribed it.
- Given an event *e*, *Eve(e)* returns a set of brokers that receive *e* to match it against the subscriptions they store.
- Event routing is a two-step process in this case: first an event *e* is sent to all brokers returned by *Eve(e)*, these brokers then match it against the subscriptions they store and notify the corresponding subscribers.
- This approach works only if the intersection between *Eve(e)* and *Sub(s)* is not empty (*mapping intersection rule*).

5.3.2 Event Flooding: Each event is broadcast from the publisher into the entire system. The implementation is straightforward but very costly. This solution has the highest message overhead with no memory overhead.

Chapter-6
IMPLEMENTATION I

6.1 INTRODUCTION

A server log is a log file (or several files) automatically created and maintained by a server of activity performed by it. A typical example is a web server log which maintains a history of page requests. The W3C maintains a standard format (the Common Log Format) for web server log files. Information about the request, including client IP address, request date/time, page requested, HTTP code, bytes served, user agent, and referrer are typically added.

These files are usually not accessible to general Internet users, only to the webmaster or other administrative person. A statistical analysis of the server log may be used to examine traffic patterns by time of day, day of week, referrer, or user agent. Efficient web site administration, adequate hosting resources and the fine tuning of sales efforts can be aided by analysis of the web server logs.

6.2 PUBLISH-SUBSCRIBE BASED WEB SERVER LOG

The log server records remote IP address of all publishers (http clients).

Publisher: http client

Subscriber: Log Server, Server Log File, etc.

Event System implemented using **node.js** (event based server side Programming Language).

6.2.1 Variables

Ip is the IP address of the user requesting the server on http,

url is the relative address that a client requests on the server,

datetime is the date and time when user requests a particular url,

log channel is the logic handle that collects the IP address and sends it to the hash structure,
Following is an example of content based publish subscribe with attributes {Ip, Url, DateTime}

6.2.2 The main logs() function:

```
var client= new publisher();
function logs(req)
 {
  var addr=req.headers["user-agent"] || "";
  client.hincrby("url",req.url,1);
  client.hincrby("ip",req.connection.remoteAddress,1);
  var datetime = new Date();
  client.publish("log",{Ip:req.connection.remoteAddress,Url:req.url, DateTime);
 }
```

6.3 ADVANTAGES OF PUBLISH-SUBSCRIBE BASED LOG SERVER

- **Monitoring-** Administrators can monitor particular IP address by subscribing to the particular IP.
- **Security-** Administrators can easily block a particular user before a malignant user issues multiple requests (say >1000) even before any request propagates to the server because of the event based nature of the server.

Chapter-7
IMPLEMENTATION II

7.1 INTRODUCTION

The internet generates millions of web pages every day. The content generated by users is mostly unstructured. Users on Social Network do not look for structured data. They publish content and encounter social content, they seldom subscribe to social content. The published content is the best speculation of user interest. But again this content is unstructured. Here a method is proposed to derive meaning out of published or subscribed content and respond to user's interest without deliberate user request. This model has applications in Blogging Systems, Advertisements and Recommendation Systems.

7.2 ALGORITHM

1. Parse published/ subscribed text to find important words or extract information,
2. When a publisher publish some content we again parse text to find important words or extract information,
3. We then compare the above two extracted information and sort by percentage similarity.

7.2.1 Implementation

Python nltk library was used to perform information extraction from text.

```
import nltk
txt = nltk.word_tokenize("Alice and her adventures in wonderland")
nltk.pos_tag(text)

[('Alice', 'NNP'), ('and', 'CC'), ('her', 'PRP'), ('adventures',
'NNS'),('in', 'IN'), ('wonderland', 'NN')]
```

The published text is split into sentences using a sentence segmenter, and each sentence is sub partitioned into words using a tokenizer. Each sentence is now tagged with part-of-speech tags, which will be quite helpful in the next step, named entity detection. In this step, we search for mentions of potentially interesting entities in each sentence.

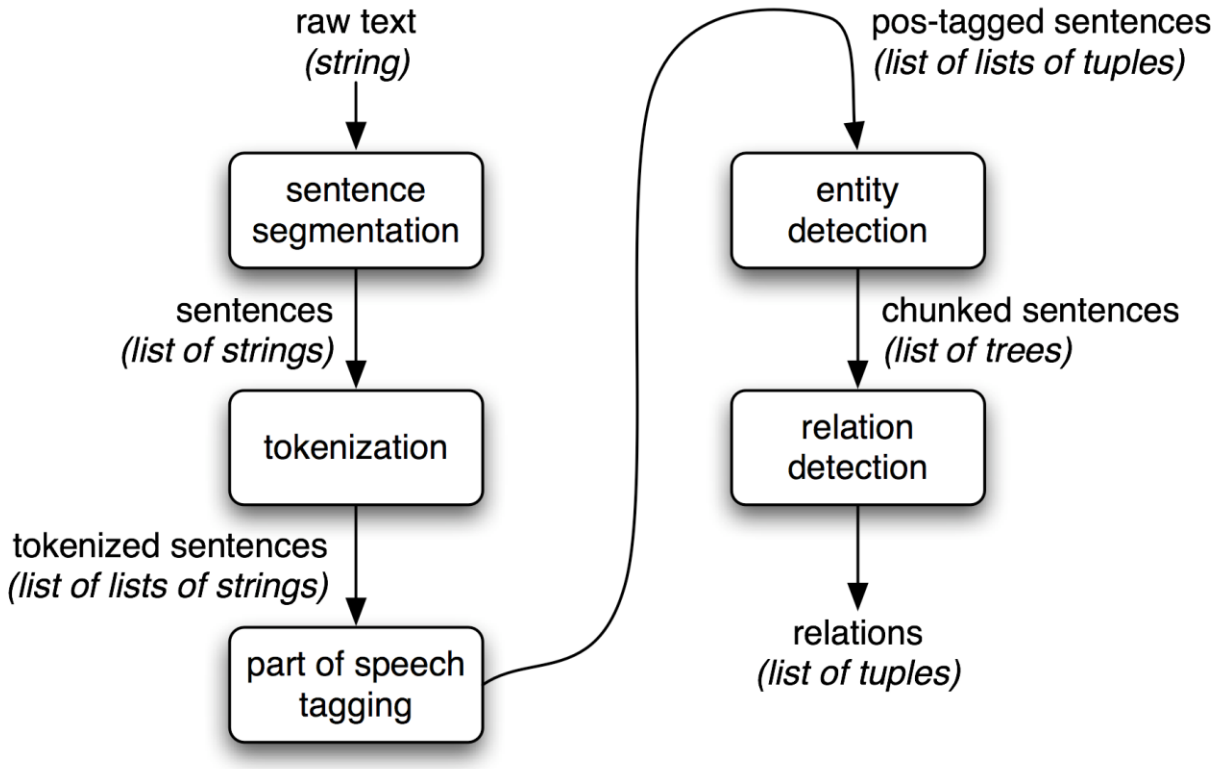

Fig. 14 Information Extraction using python nltk

http://nltk.googlecode.com/svn/trunk/doc/book/ch07.html

To perform first 3 tasks of sentence segmentation, tokenization and part of speech tagging following method is used:

```
def txt_preprocess(published_text):
...     sens = nltk.sent_tokenize(published_text)[1]
...     sens = [nltk.word_tokenize(sent) for sent in sens] [2]
...     sens = [nltk.pos_tag(sent) for sent in sens]
```

Next, in named entity detection, we segment and label the entities that might participate in interesting relations with one another. Typically, these will be definite noun phrases such as the knights who say "ni", or proper names such as Pandit Gangadhar Vidyadhar Mayadhar Omkarnath Shastri. In some tasks it is useful to also consider indefinite nouns or noun chunks, such as every student or cats, and these do not necessarily refer to entities in the same way as definite NPs and proper names.

Entity recognition is often performed using chunkers, which segment multi-token sequences, and label them with the appropriate entity type. Common entity types include ORGANIZATION, PERSON, LOCATION, DATE, TIME, MONEY, and GPE (geo-political entity).

Chunkers can be constructed using rule-based systems, such as the RegexpParser class provided by NLTK; or using machine learning techniques. In either case, part-of-speech tags are often a very important feature when searching for chunks.

Relation extraction can be performed using either rule-based systems which typically look for specific patterns in the text that connect entities and the intervening words; or using machine-learning systems like Naïve Bayesian or Maxent Classifier in python nltk which typically attempt to learn such patterns automatically from a training corpus.

7.3 SOFTWARES USED

nltk (NLP library for python)
node.js (server side)
node.Natural (NLP library for node)

Chapter-8
CONCLUSION

6.1 Conclusion:

This thesis work includes study of Distributed Systems, Publish Subscribe model of communication and its various forms. A scheme is proposed to present users with updates that they might be interested into based on their published content or subscribed text both of which are unstructured. Natural language processing is used to find the best data that a user might be interested in using Information Extraction using NLTK library in python.

6.2 Scope for Future Work:

Algorithm proposed assumes only for centralized Event Brokers but can be applied in distributed context as well. Distributed unsupervised methods to predict user interest and present them with results may be implemented.

REFERENCES

[1.] Tanenbaum, Andrew S., and Maarten Van Steen. *Distributed systems*. Vol. 2. Prentice Hall, 2002.

[2.] The Many Faces of Publish/Subscribe PATRICK TH. EUGSTER, Swiss Federal Institute of Technology, Lausanne**,** PASCAL A. FELBER**,** Institut Eur´ecom**,** RACHID GUERRAOUI**,** Swiss Federal Institute of Technology, Lausanne AND ANNE-MARIE KERMARREC Microsoft Research

[3.] Baldoni, R., Contenti, M., Virgillito, A.: The Evolution of Publish/Subscribe Systems. In Andŕe Schiper and Alexander A. Shvartsman and Hakim Weathe rspoon and Ben Y. Zhao,ed.: Future Trends in Distributed Computing, Research and Position Papers.

[4.] Rajkumar, Ragunathan, Michael Gagliardi, and Lui Sha. "The real-time publisher/subscriber inter-process communication model for distributed real-time systems: design and implementation."*Real-Time Technology and Applications Symposium, 1995. Proceedings*. IEEE, 1995.

[5.] Gupta, Abhishek, et al. "Meghdoot: content-based publish/subscribe over P2P networks."*Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*. Springer-Verlag New York, Inc., 2004

[6.] Petri Jokela, András Zahemszky, Christian Esteve Rothenberg, Somaya Arianfar, and Pekka Nikander. Ericsson Research, NomadicLab, Finland {petri.jokela, andras.zahemszky, somaya.arianfar, pekka.nikander}@ericsson.com University of Campinas (UNICAMP), Brazil chesteve@dca.fee.unicamp.br

[7.] http://pages.cs.wisc.edu/~sschang/firewall/gcb/mechanism.htm

[8.] http://docs.oracle.com/cd/B19306_01/server.102/b14231/ds_concepts.htm

[9.] http://en.wikipedia.org/wiki/Distributed_object Author: Stanislav Tvaruzek

[10.] Python nltk library http://www.nltk.org/book/